Санкт–Петербургский государственный университет

ЛУКЬЯНЧИКОВ Константин Анатольевич

Выпускная квалификационная работа

Генерация музыки с помощью нейронных сетей

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2016 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Математическое и программное обеспечение вычислительных машин»

Научный руководитель:
доцент, кафедра технологии програмамирования,
к.т.н. Блеканов Иван Станиславович

Рецензент:
доцент, кафедра компьютерных технологий
и систем, к.ф.-м.н. Погожев Сергей Владимирович

Санкт-Петербург
2020 г.

Содержание

Введение

Актуальность

В наши дни искусственный интеллект активно используется для улучшения многих аспектов повседневной жизни людей. Он дает нам рекомендации по предметам, которые мы хотим приобрести, переводит текст с одного языка на другой, подбирает музыкальный плейлист на основе наших предпочтений, а также выполняет еще множество полезных действий, которые делают нашу жизнь лучше и проще.

Нейронные сети — один из видов машинного обучения. В основе их работы лежат способы функционирования биологических нейронный сетей. Модели, построенные с их помощью являются очень мощными инструментами для решения множества разных задач. Однако, порой, чтобы обучить модель требуется большое количество ресурсов в виде времени и вычислительных мощностей.

Генеративное моделирование — одна из самых обсуждаемых тем в области искусственного интеллекта. Машины можно научить не только рисовать и писать текст, но и сочинять музыку, для этого достаточно использовать актуальные примеры генеративных моделей глубокого обучения.

Создание машиной реалистично звучащей музыки является сложной и в то же время интересной проблемой. Существует две основные ветви разработки в этой области. Первая нацелена на решение задачи автоматической генерации музыкальных композиций без участия человека. Вторая же на создание программного обеспечения, которое предоставляет интерфейс для взаимодействия с человеком, и способна помогать при создании музыкального произведения.

Большинство людей получают удовольствие от прослушивания музыки, и если есть какой-то способ генерировать музыку автоматически, особенно достойного качества, то это большой скачок в мире музыкальной индустрии.

В данной работе поставлена задача реализовать и обучить модель, решающую задачу автоматической генерации музыки.

Цель работы

Цель выпускной квалификационной работы является реализация метода автоматической генерации музыки с помощью нейронных сетей.

Постановка задачи

Для достижения цели необходимо:

1. Проанализировать существующие методы и подходы к решению задачи генерации музыки.
2. Описать и реальзовать архитектуру, способную генерировать музыку.
3. Путем варьирования параметров, получить несколько моделей для сравнительного анализа.
4. Обучить различные модели и сравнить их показатели.
5. Получить музкальные фрагменты, сгенерированные лучшей моделью.

Практическая значимость

Практическая значимость данного иследования крайне высока. Развитие области автоматической генерации музыки вносит вклад в жизни не только музыкантов, но и людей, для которых музыка не является основным видом деятельности.

Композиторам более не придется заниматься утомительными вычислениями и они смогут всецело посвятить себя творчеству: исследованию и улучшению музыкальной формы, которую создал для них компьютер.

Для игровой и киноиндустрии процесс создания сопровождающих композиций упростится с точки зрения временных и финансовых затрат.

У обычных людей будет возможность не выходя из дома и в любой момент времени услышать свеже сгенерированный музыкальный фрагмент, реализованный на любом существующем музыкальном инструменте, или при желании даже разыгранный целым виртуальным оркестром!

На данном этапе развития этой области уже существуют нейронный сети, которые способны генерировать музыку в стиле Моцарта или Баха. Что означает для нас наличие возможности услышать потенциально возможные композиции многих творцов прошлого и даже тех, кто находился у самых истоков современной музыкальной теории.

Программные средства использованные в работе

В качестве языка программирования был выбран Python, так как он лучше других языков подходит для задач машинного обучения в силу своей гибкости и поддержки большого количества полезных библиотек. Для обработки музыкальных данных было решено выбрать библиотеку Music21, так как она хорошо справляется с обработкой MIDI файлов. Для реализации нейросетевой части программы, была выбрана библиотека Keras в силу своей простоты и компактности.

Глава 1. Используемые инструменты и архитектуры

1.1 Рекуррентные нейронные сети

Традиционные нейронные сети не обладают свойством памяти, они не способны использовать рассуждения о предыдущих событиях, чтобы получить информацию о последующих.

Решить эту проблемы помогают рекуррентые нейронные сети (Recurrent Neural Networks, RNN). Это сети, содержащие обратные связи и позволяющие сохранять информацию. В [5] подробно описывается работа RNN:

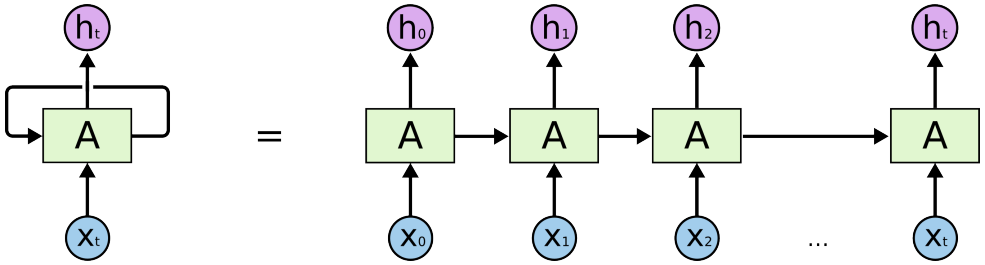

Рис. 1: Рекуррентная нейронная сеть в развертке.

RNN обрабатывает последовательность, перебирая ее элементы и сохраняя состояние, полученное при обработке предыдущих элементов. Фактически, RNN — это разновидность нейронной сети, имеющей внутренний цикл (Рис. 1).

На вход RNN принимает последовательность векторов и учитывая текущее состояние и входные признаки, конструирует выходной результат для момента времени t. Этот выходной результат затем будет сохраняться во внутреннем состоянии как подготовка к следующей итерации.

На каждом шаге обучения t значение скрытого слоя (состояния) рекуррентной нейронной сети $h_t \in R^m$ вычисляется по формуле

$$h_t = f(Wx_t + Uh_{t-1} + b_h),$$

где $x_t \in R^n$ — входной вектор в момент времени t, $W \in R^{m \times n}$, $U \in R^{m \times m}$, $b_h \in R^m$ — обучаемые параметры сети, f — некоторая функция активации (например, поэлементно применяемая к компонентам вектора сигмоида).

Функция f преобразует входные данные и состояние и параметризуется двумя матрицами W, U и вектором смещений.

Любая рекуррентная нейросеть имеет форму цепочки повторяющихся модулей нейронной сети (Рис. 2). В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации tanh (гиперболический тангенс).

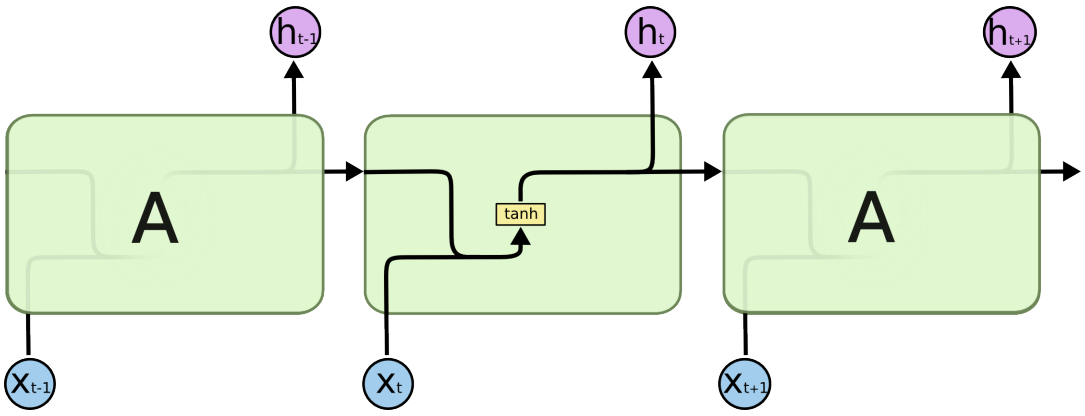

Рис. 2: Повторяющийся модуль в стандартной RNN состоит из одного слоя.

Такие сети хорошо справляются с решением задач языкового моделирования, перевода, генерации последовательностей и т.д., то есть там, где нужно связывать информацию о текущем и предыдущих моментах времени.

Но у классической модели рекуррсентной сети есть проблемы с обработкой долговременных зависимостей. То есть по мере увеличения расстояния между свежей информацией и информацией из более отдаленного прошлого, она теряет способность их запоминать и связывать.

1.2 LSTM сети

Сети LSTM (Long Short-Term Memory) — особая модификация архитектуры рекуррентных нейронных сетей, была представлена Зеппом Хохрайтером и Юргеном Шмидхубером в 1997 году [3]. В отличие от простых RNN, которым не хватает долгосрочной памяти, LSTM действует, как конвейер, который может переносить информацию вперед через значительные промежутки времени.

Есть несколько особенностей LSTM сетей, которые делают их подходящими для обработки последовательностей, в частности, последовательностей нот и аккордов :

- LSTM — это рекуррентная нейронная сеть, которая обрабатывает информацию последовательно во времени.
- Ее архитектура позволяет находить некоторые логические паттерны.
- Кроме того, способность запоминать и повторно использовать входные данные в течении длительного периода времени, воспроизводя своего рода долговременную память, чрезвычайно полезна в задачах обрабтки и генерации последовательностей.

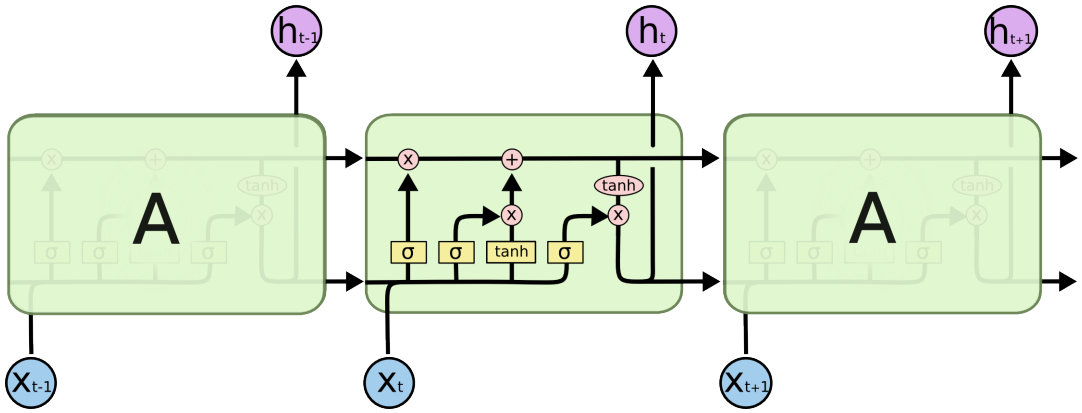

Рис. 3: Повторяющийся модуль в LSTM сети состоит из четырех взаимодействующих слоев.

Ключевой компонент LSTM — это состояние ячейки — линия, проходящая по верхней части схемы (Рис. 3, 4). Оно проходит напрямую через всю цепочку, и подвержено лишь нескольким линейным преобразованиям. Таким образом информация может легко проходить по ней, не подвергаясь изменениям.

Однако, LSTM имеет возможность удалять некоторую информацию из ячейки состояния и записывать новую, этот процесс регулируется специальными фильтрами (gates), которые обеспечивают выполнение трех задач

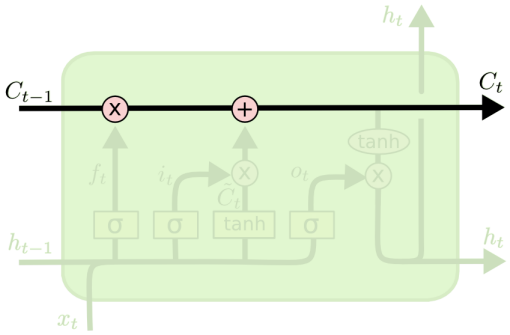

Рис. 4: Состояние ячейки (cell state) LSTM.

по управлению информационным потоком: забывание предыдущей информации, ввод и вывод. Таким образом сеть определяет какие аспекты нового состояния релевантны для нового входа, какие аспекты из предыдущих состояний должны быть забыты, и какая часть состояния переносится вперед.

Далее подобнее рассмотрим структуру ячейки LSTM сети.

1.2.1 Forget gate

Первый шаг — определить, какую информация можно выбросить из состояния ячейки (забыть). Это решение принимает слой, называемый forget gate layer. Основываясь на h_{t-1} (выход предыдущей ячейки) и x_t (вход текущей ячейки), он возращает число от 0 до 1 для каждого числа из состояния ячейки C_{t-1}, где 1 означает «полностью сохранить», а 0 – «полностью выбросить» (Рис. 5).

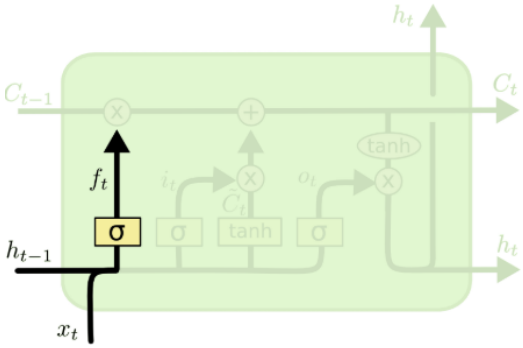

Рис. 5: Forget layer.

Значение f_t вычисляется по формуле

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f),$$

где σ — сигмоидная функция активации, W_f — матрица весов данного слоя, b_f — вектор смещений данного слоя. Осталось заменить старое состояние C_{t-1} на новое состояние C_t, умножая старое состояние C_{t-1} на f_t, удаляя то, что мы решили забыть.

1.2.2 Input gate

Второй шаг — решить, какую новую информацию нужно записать в состояние ячейки (обновить состояние). Этот этап делиться на две части (Рис. 6):

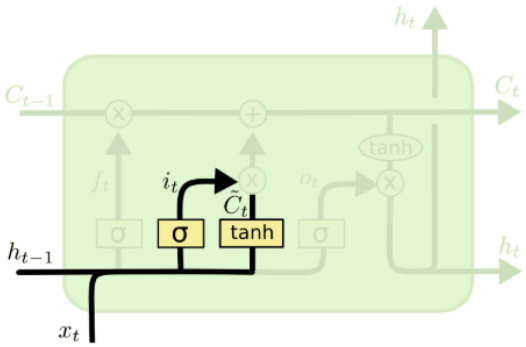

Рис. 6: Input layer.

1. Сначала h_{t-1} и x_t проходят через сигмоидальный слой под названием input gate. Это позволяет определить какие значения должны быть обновлены, путем преобразования значений в числа $\{0, \ldots, 1\}$ по формуле:

   $$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$

   где σ — сигмоидная функция активации, W_i — матрица весов данного слоя для входных значений, а b_i — вектор смещений данного слоя для входных значений.

2. Затем h_{t-1} и x_t пропускаются через tanh — слой для определения вектора новых значений — кандидатов $\tilde{C}_t$, которые можно добавить в

состояние ячейки по формуле

$$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C),$$

где $tanh$ — функция активации гиперболический тангенс, W_C — матрица весов данного слоя для значений кандидатов, b_C — вектор смещений данного слоя для значений кандидатов.

Далее обновляем C_t по формуле (Рис. 7):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C}_t.$$

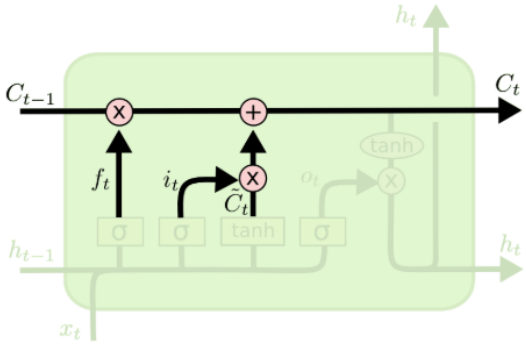

Рис. 7: Обновление состояния C ячейки в момент времени t.

1.2.3 Output gate

Наконец, определяется выходное значение h_t ячейки LSTM. Решение зависит от выходного значения предыдущей ячейки h_{t-1}, входного значения текущей ячейки x_t и обновленного значения ячейки состояния C_t (Рис. 8).

Данный этап разбивается на два шага:

1. Для начала h_{t-1} и x_t проходят через сигмоидный слой под названием output gate. Это позволяет определить какие значения ячейки состояния должны быть переданы следующей ячейке, путем преобразования значений в числа $\{0, \ldots, 1\}$ по формуле

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o),$$

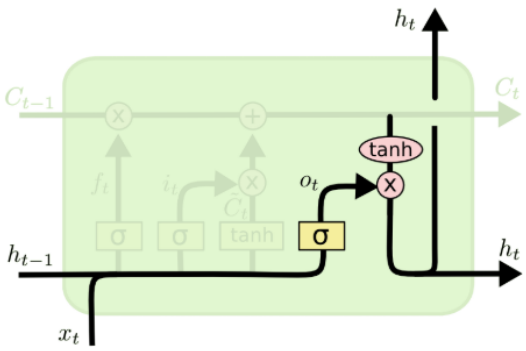

Рис. 8: Output gate.

где σ — сигмоидная функция активации, W_o — матрица весов данного слоя, а b_o — вектор смещений данного слоя для входных значений.

2. Затем значения ячейки состояния C_t проходят через функцию $tanh$ для преобразования их в числа $\{-1, \ldots, 1\}$ и перемножаются со значениями o_t, вычисленными на слое output:

$$h_t = o_t \cdot tanh(C_t).$$

1.3 Двунаправленные рекуррентные сети

Двунаправленные рекуррентные сети (Bidirectional RNN) — разновидность RNN, предложенная в 1997 году [10], зависящая не только от предыдущих элементов последовательности, но и от следующих (Рис. 9). На практике такая сеть состоит из:

- Одна реккурентная сеть, движущаяся вперед во времени (forward RNN). Она начинает свое движение с начала последовательности.
- Вторая реккурентная сеть, движущаяся назад во времени (backward RNN). Она начинает свое движение с конца последовательности.

Выходом такой сети является вектор y_t, который представляет собой конкатенацию двух векторов:

1. Выход скрытого слоя forward RNN s_t^f в момент времени t.
2. Выход скрытого слоя backward RNN s_{i-t+1}^b в момент времени $i-t+1$.

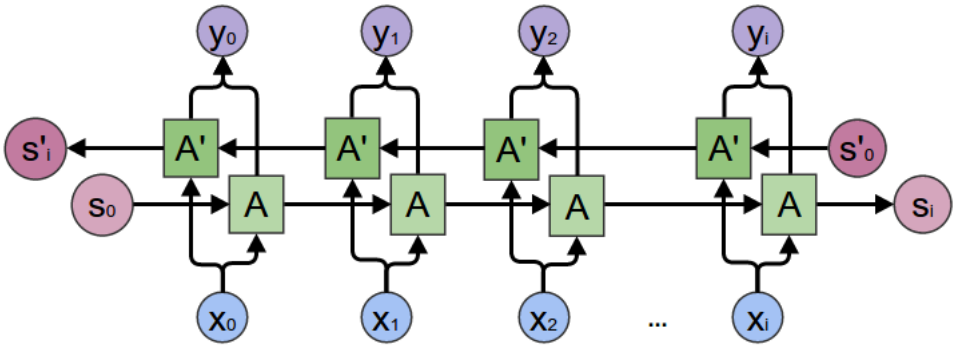

Рис. 9: Bidirectional RNN.

1.4 Механизм внимания

Механизм внимания (Attention mechanism, Attention) — подход в современном машинном обучении, который предназначен для выделения некоторой части из входных данных для более делальной обработки. Впервые был предложен в 2015 году [11]. Attention обычно используется в связке с LSTM и применяется непосредственно после рекуррентного слоя. Существуют несколько разновидностей данного механизма, в этой работе остановимся на одном из них, который был рассмотрен в [12].

Обычно выход LSTM состоит из вектора h_t, который был вычислен последней ячейкой. Но бывают ситуации, когда невозможно представить всю информацию о входной последовательности в одном векторе фиксированной длины. Другими словами, порой сложно делать хорошие предсказания исходя лишь из одного вектора.

Как было сказано ранее, Attention позволяет выделить важные векторы из входной последовательности. С этим механизмом, мы даем модели возможность *обратить внимание* на разные части входной последовательности в разные моменты времени. Для этого необходимо давать на вход этому слою не только h_t, но и все последовательности скрытых состояний LSTM $\{h_1, h_2, ..., h_n\}$.

Слой Attention представлен матрицей A, позволяющей уловить связь одного члена последовательности с его соседями. Элемент матрицы $a_{t,t'} \in A$ устанавливает связь между скрытыми состояниями h_t и $h_{t'}$ членов последовательности x_t и x'_t в моменты времени t и t' соответственно. Расчет

$a_{t,t'}$ происходит по формуле

$$a_{t,t'} = \sigma(W_a \cdot g_{t,t'} + b_a),$$

$$g_{t,t'} = tanh(W_g \cdot h_t + W_{g'} \cdot h_{t'} + b_g),$$

где σ — поэлементная сигмоидная функция, W_g и $W_{g'}$ — матрицы весов, соответствующие скрытым состояниям h_t и $h_{t'}$ соответственно, W_a — матрица весов их нелинейной комбинации, а b_g и b_a векторы смещений.

Результат применения такого механизма внимания — последовательность векторов $l = \{l_1, \ldots, l_n\}$, где l_t — член последовательности в момент времени t, полученный суммой произведений скрытых состояний $h_{t'}$ в моменты времени t' и их связей со скрытым состоянием h_t текущего члена последовательности $a_{t,t'}$

$$l_t = \sum_{t'=1}^{n} a_{t,t'} \cdot h_{t'}.$$

Другими словами, l_t контролирует *сколько внимания нужно уделить* члену входной последовательности в любой момент времени, основываясь на контексте его соседей.

1.5 Сверточные сети

Сверточные нейронные сети (CNN) обычно применяются для обработки изображений. Сеть старается подражать принципам человеческого зрения, используя при этом математические операции свертки. Впервые сверточные нейронные сети были применены для обработки рукописных символов в 1998 году [13]. Наилучшие результаты CNN показывают в области распознования изображений, но применяются не только в ней.

В основе CNN лежат три процесса, которые обеспечивают достижение инвариантности к переносу, незначительным искажениям и масштабированию:

1. Локальные рецептивные поля, которые обеспечивают локальную связность нейронов.

2. Общие синаптические коэффициенты, обеспечивающие выделение некоторых признаков в любом месте изображения и уменьшают общее количество весовых коэффициентов.

3. Иерархическая организация с пространственными подвыборками (локальное усреднение, приводящее к понижению чувствительности к смещению и деформациям).

Основная идея CNN:

- Матрица (ядро, или фильтер) скользит через все изображение (представленное в матричном виде);
- На каждой позиции:
  - Считается скалярное произведение фильтра и соответствующей части входной матрицы;
  - Значения суммируются;
- Результат — новая матрица (карта признаков), составленная из сумм, полученных при скольжении ядра по входной матрице.

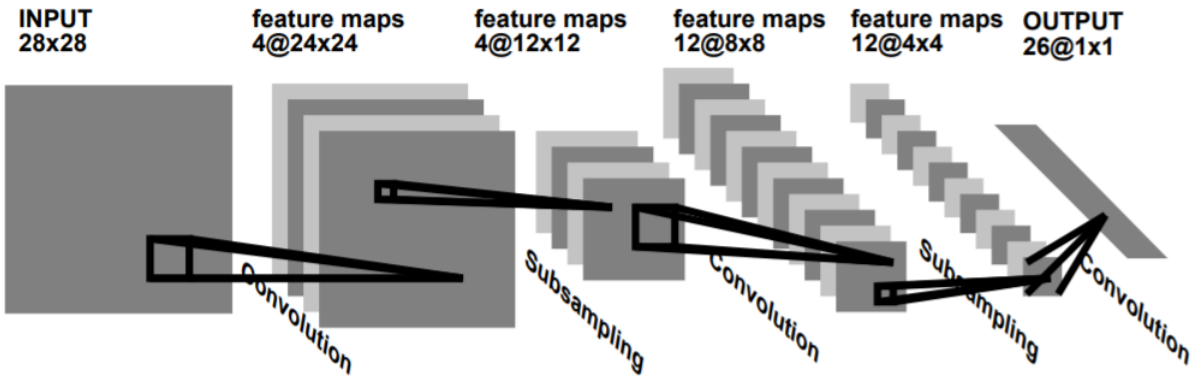

Рис. 10: Сверточная нейронная сеть [12].

Например, если на входе имеется двумерное изображение в матричном виде A и ядро B, то операция свертки происходит по формуле

$$s(i,j) = \sum_m \sum_n A(m,n)B(i-m, j-n).$$

Размер карты признаков определяется тремя параметрами:

1. *Depth* - количество использованных ядер;
2. *Stride* - шаг с которым скользила матрица;
3. *Padding* - параметр, отвечающий за дополнение исходной матрицы по краям.

CNN состоит из разных видов слоев (Рис. 10): сверточные, подвыборочные слои, цель которых уменьшение размерности карт предыдущего слоя, а также полносвязные слои. Первые два формируют входной вектор признаков, чередуясь друг с другом. Третий отвечает за классификацию, моделируя сложную нелинейную функцию, оптимизирую которую, улучшается качество распознования.

1.6 Автокодировщик

Автокодировщик (Autoencoder) — это архитектура нейронных сетей, позволяющая применять обучение без учителя. Отличительная черта этой модели в том, что количество нейронов на входном слое равно количеству нейронов на выходном. Одними из первых, кто применил автокодировщики в глубоком обучении были Джеффри Хинтон и Расс Салахутдинов в 2006 году [14].

Простейшая версия автокодировщика (Рис. 11) включает в себя только входной, промежуточный и выходной слои. Главной задачей при обучении такой сети — получить на выходном слое данные максимально похожие на те, что были на входном. Однако, если размер промежуточного слоя совпадает с размером входного и выходного слоев, то задача является тривиальной. Поэтому, чаще всего промежуточный слой ограничивается меньшим числом нейронов, чем на выходном и выходном слоях и разреженностью (когда большинство элементов имеют нулевые значения).

Такие органичения заставляют нейросеть искать общие признаки во входных данных и сжимать данные на промежуточном слое (encoding), а также восстанавливать исходные данные из сжатого представления (decoding).

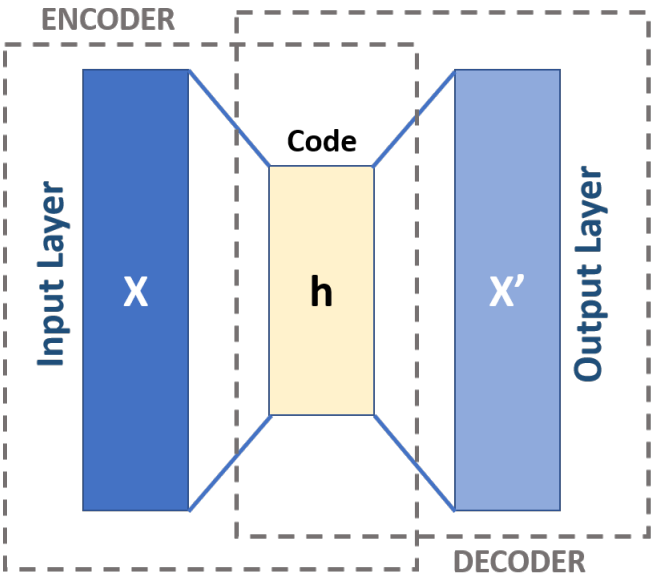

Рис. 11: Схематичная архитектура автокодировщика.

- Таким образом сеть разделяется на две принципиально разные части:

$$\textit{Кодировщик: } h = f(x),$$
$$\textit{Декодировщик: } x' = g(h).$$

- Слои описываются преобразованиями следующего вида

$$h = f(x) = s_f(W_h * x + b_h), \quad x' = g(h) = s_g(W_{x'} * h + b_{x'})),$$

где s_f — функции активации (sigmoid, tanh, relu) промежуточного слоя, W_h — матрица весов промежуточного слоя и b_h — вектов смещений промежуточного слоя. А параметры s_g, $W_{x'}$, $b_{x'}$ соответсвуют выходному слою.

- Обучение сводится к задаче минимизации функции восстановления (как например: *mean squared error*)

$$L(x, x') = \|x - x'\|^2 = \|x - \sigma'(W_{x'}(\sigma(W_h + b_h)) + b_{x'})\|^2 .$$

1.7 Вариационный автокодировщик

Вариационные автокодировщики (Variational Autoencoder, VAE), представленные одновременно Дидериком Кингма и Максом Веллингом в декабре 2013 года [15] и Данило Йименезом Резенде, Шакиром Мохамедом и Дааном Вирстрой в январе 2014 года [16], являются автокодировщиками, основанными на вариационном выводе.

На практике фундаментальная проблема с обычными автокодировщиками заключается в том, что скрытое пространство, в которое они преобразуют свои входные данные и где лежат их закодированные векторы, может быть не непрерывным и не структурированным. Поэтому автокодировщики не очень хороши как инструменты сжатия и генерации.

VAE накладывают дополнительные ограничения на кодирумые данные. Они составляют статистическую модель, которая соотносит набор входных данных с набором скрытых переменных. Поэтому вместо того, чтобы нейросеть конструировала произвольную функцию отображения, она изучает параметры распределения вероятностей, моделирующие данные. Отбирая точки из этого распределения, можно генерировать новые образцы, основанные на входных данных.

Вместо сжатия в фиксированный вектор в скрытом пространстве, кодировщик превращает входной вектор в параметры статистического распределения: μ (математическое ожидание) и σ (среднеквадратическое отклонение). Таким образом, предполагается, что входной ветор был сгенерирован статистическим процессом и при кодировании и декодировании необходимо учитывать случайную составляющую этого процесса. VAE использует μ и σ как параметры для случайного выбора одного элемента из разпределения и декодирует этот элемент обратно в оригинальный вектор. Стохастичность этого процесса повышает надежность и заставляет скрытое пространство кодировать значимые представления. Т.е. каждая точка, выбранная в скрытом пространстве, декодируется в допустимый вывод.

С технической точки зрения этот процесс выглядит так:

1. Кодировщик превращает выборки из входных данных в два параметра в скрытом пространстве (μ и σ).

2. Из скрытого распределения выбирается произвольная точка z для генерации входного вектора, как $z = \mu + \sigma \cdot \varepsilon$, где $\varepsilon \sim N(0, 1)$.

3. Декодер отображает эту точку из скрытого пространства обратно в оригинальный вектор.

Так как ε — случайный вектор, процесс гарантирует, что каждая точка, близкая к скрытым представлениям входных данных, будет декодирована в нечто похожее на входные данные.

Функция потерь VAE состоит из двух слагаемых:

1. Потери восстановления (reconstruction loss), заставляет декодированные образцы совпадать с теми, что были поданы на вход.

2. Потери регуляризации (regularization loss), помогает извлекать непрерывные, хорошо структурированные пространства и ослабляет эффект переобучения.

Подробнее устройство вариационного автокодировщика было описано в статье 2016 года «Tutorial on Variational Autoencoders» [17].

1.8 Генеративно-состязательные сети

Генеративно-состязательные сети (GAN) впервые были представлены Яном Гудфеллоу и его коллегами в 2014 году [18]. Идея этой архитектуры заключается в том, чтобы тренировать две нейронные сети одновременно (Рис. 12). Каждая из них отвечает за свою задачу:

- *Генератор* — получает на входе случайный вектор (в начале — шум) и декодирует его в искусственный сэмпл.

- *Дискриминатор* — получет на вход сэмпл (настоящий или созданный генератором) и определяет его подлинность.

Формально, имеется множество примеров X и его подмножество подлинных примеров $Y \subset X$. Таким образом, на множестве X определено вероятностное распределение $P_{data}(x)$. Задача состоит в том, чтобы отыскать две функции:

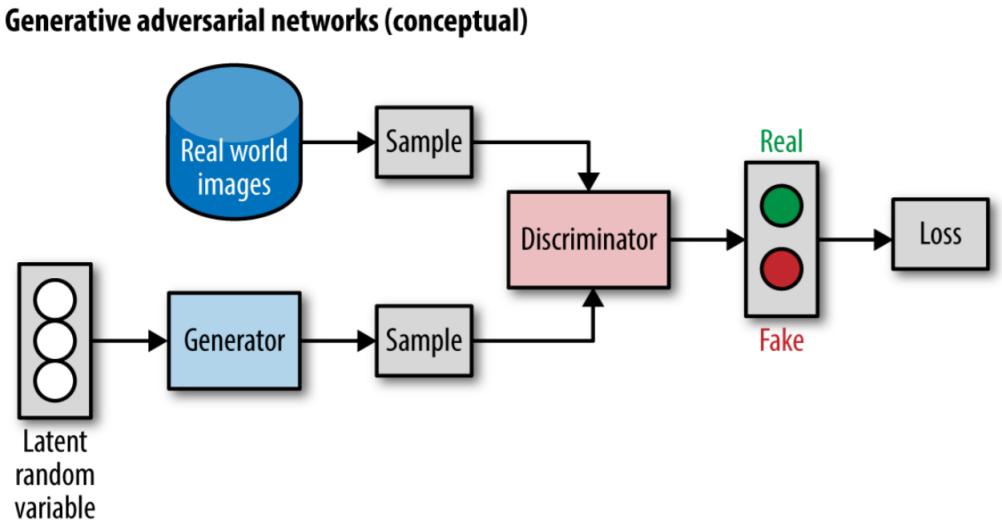

Рис. 12: Архитектура генеративно — состязательных сетей [19].

1. $G(z, \theta_g)$, где z — случайный шум, заданный распределением $P_z(z)$, θ_g — тренеруемые параметры генератора. На выходе функция G должна выдавать пример из множества X. Следовательно, G задает новое распределение $P_g(x)$.

2. $D(x, \theta_d)$, где $x \in X$, а θ_d — тренеруемые параметры дискриминатора. Значение $D(x)$ — вещественное число, которое отражает вероятность того, что $x \in Y$, а не создано генератором.

Во время тренировки ищутся такие параметры θ_d, чтобы максимизировать вероятность правильного разделения функцией D настоящих примеров и примеров, созданных функцией G. И параметры θ_g, чтобы минимизировать функцию $\log(1 - D(G(z)))$.

Задачу можно рассмотреть, как minimax игру двух игроков, где $V(G, D)$ — целевая функция

$$\min_G \max_D V(D, G) = \mathbb{E}_{data}(\log D(x))) + \mathbb{E}_z(\log(1 - D(G(z))))$$

При обучении, алгоритм попеременно оптимизирует функцию D при фиксированной G, а затем фиксируя функцию D, оптимизирует функцию G. При этом возникает проблема, что на начальном этапе функция G плохо

справляется с генерацией примеров, похожих на настоящие и, следовательно, D с большой уверенностью назначает им очень маленькие вероятности. Таким образом, значение $\log(1-D(G(z)))$ близко к 0, соотвтственно обучение параметров G происходит медленно. Исходя из этого, авторы предлагают максимизировать $\log(D(G(z)))$, вместо минимизации $\log(1-D(G(z)))$. Такая задача является эквивалентной, при этом улучшаются характеристики градиентов.

Глава 2. Данные

В этой главе мы рассмотрим этапы и способы работы с данными касательно задачи генерации музыки.

Предобработка данных является один из важнейших этапов в задачах генерации, от нее зависит как будет протекать процесс обучения, а также качество того, что мы получим в процессе генерации.

2.1 Тип данных

В области генерации музыки методами машинного обучения существует выбор между двумя типами данных:

- Аудио формат;
- Символьный формат.

От выбора зависит, будут наши данные непрерывные (аудио) или же дискретные (символьный). Первый формируется на основе знаний об обработке сигналов, а второй на основе знаний о музыке и ее теории. Однако, это не особо влияет на архитектуру сетей. Действительно, на уровне обработки глубокой нейронной сети данных, первоначальное различие между звуковом и символьным представлениями сводится к минимуму, так как рассматриваются только числовые значения и операции над ними.

Стоит отметить, что большинство современных моделей глубокого обучения для генерации музыки используют символьных формат.

Рассмотрим подробнее каждый из них.

2.1.1 Аудио формат

Главным объектом исследования в этой области является аудио сигнал. Существует несколько форм представления таких сигналов:

- Форма волны (Waveform);
- Спектрограмма (Spectrogram);
- Хромограмма (Chromagram).

Форма волны — прямое представление необработанного аудио сигнала. Визуально показано на (Рис. 13). Ось x соответствует времени, а ось y - амплитуде волны.

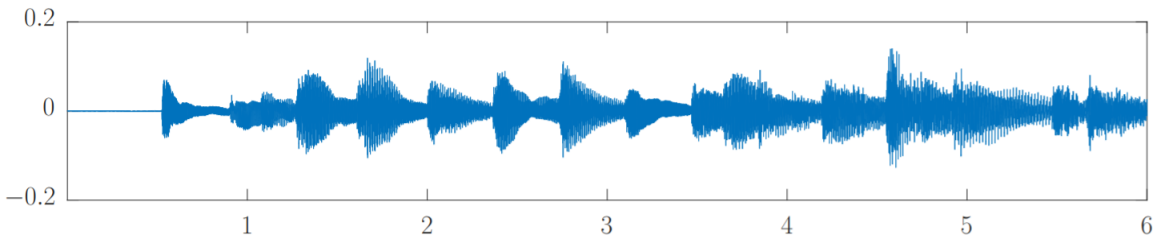

Рис. 13: Пример формы волны из [20].

Плюсом использования такого представления является то, что данные не подвержены начальной обработке и содержат миксимальное количество информации о самом аудио сигнале. Архитектуры использующие такой подход называются *end-to-end*. Недостаток заключается в вычислительной нагрузке: низкоуровневый необработанный сигнал требует больших затрат памяти и вычислительных ресурсов для обработки.

Спектрограмма — один из видов представления обработанного аудио сигнала. Визуальный пример (Рис. 14):

Наиболие распространенным представлением спектрограммы является двумерная диаграмма. По оси x — время в секундах, по оси y — частота в кГц, третья ось - цвет, отвечающая за амплитуду на определенной частоте в конкретный момент времени в dBFS.

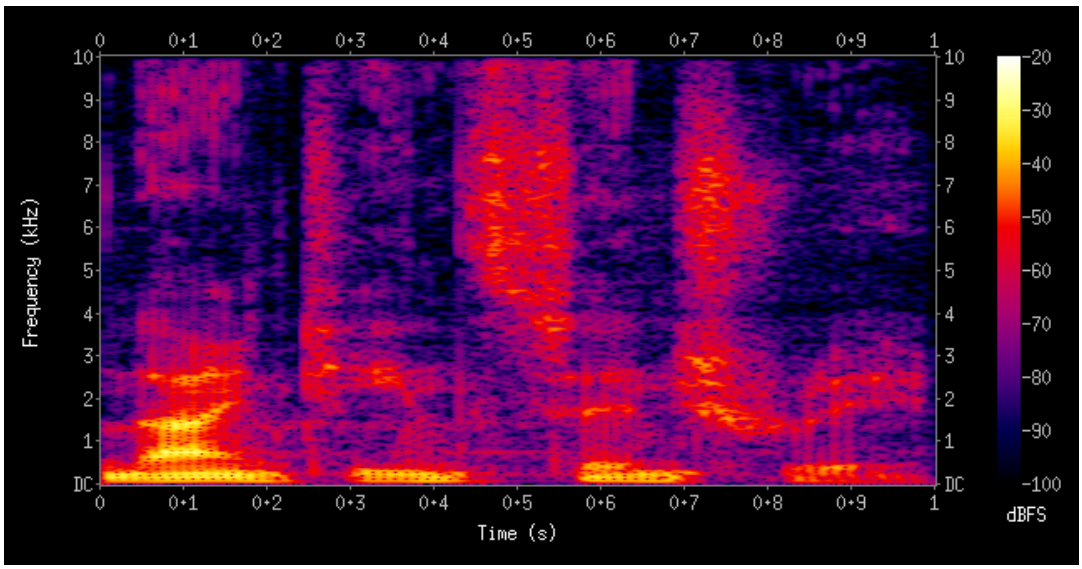

Рис. 14: Пример спектрограммы мужского голоса, взятый из https://en.wikipedia.org/wiki/Spectrogram.

Хромограмма — еще один вид представления обработанного аудио сигнала. Является вариацией спектрограммы, дискретизированная относительно времени и независима от октавы. Ограничена классами нот. Визуальный пример (Рис. 15).

По оси x — время в секундах для всех примеров (a - d). По оси y: для (a) — нота, для (b и d) — класс ноты, для (c) — амплитуда третья. Для хромограмм (b и d) третья ось — цвет, отвечающий за амплитуду на определенной частоте в конкретный момент времени в dBFS.

2.1.2 Символьный формат

Символьные представления связаны с такими понятиями, как высота, длительность и громкость.

- Высота (pitch) может быть определена:
  - Частотой в Гц;
  - Вертикальной позицией в представлении;
  - Системой нотации, совмещающей название ноты (A, $A\sharp$ и т.д.) и номер октавы (A_4 соответствует $A440$, где 440 - частота в Гц).

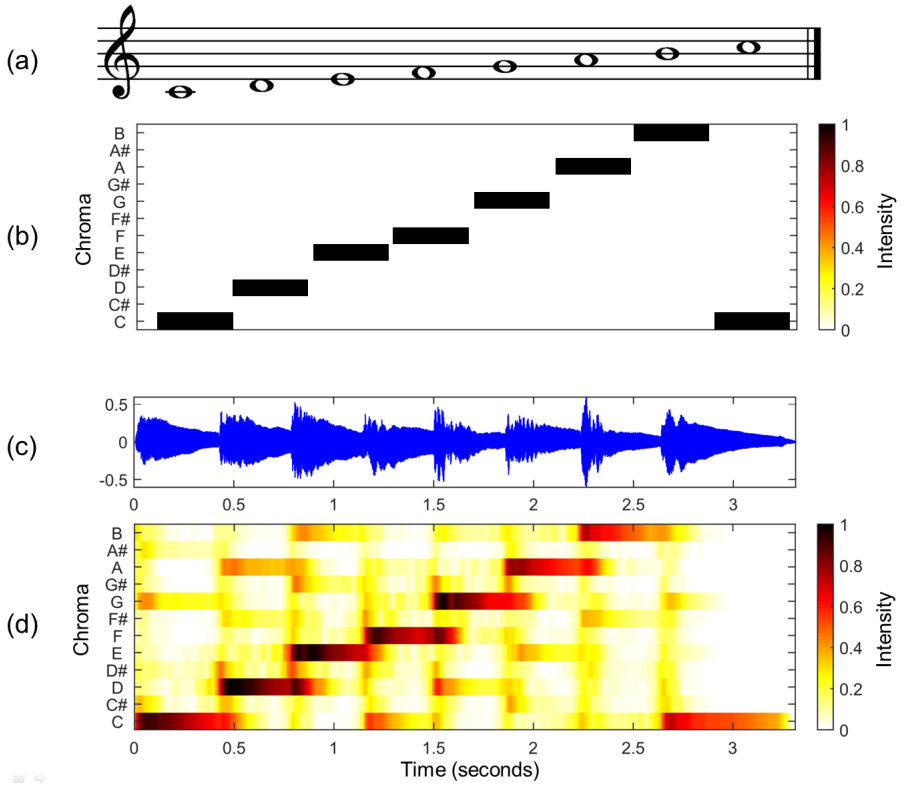

Рис. 15: Пример хромограммы гаммы До-мажор. (a) Музыкальная нотация. (b) Хромограмма нотации. (c) Аудио волна, записанная на фортепиано. (d) Хромограмма записанного аудио. Пример взят из https://en.wikipedia.org/wiki/Chroma_feature.

- Длительность (duration), может быть определена:
  - Абсолютным значением в милисекундах;
  - Относительным значением (длительности в нотной грамоте).
- Громкость (velocity), может быть определена:
  - Абсолютным значением в дБ;
  - Относительным значением (громкости в нотной грамоте).

2.1.3 MIDI

MIDI интерфейс позволяет единообразно кодировать в цифровой форме такие данные как нажатие клавиш, настройку громкости и других акустических параметров, выбор тембра, темпа, тональности и др., с точной привязкой во времени.

1. Note on —- указывает на то, что нажата нота. Содержит в себе:

   - channel number, номер канала который указывает на инструмент, определенный целым числом $\{0, 1, \ldots, 15\}$;
   - MIDI note number, указывает на высоту ноты $\{0, 1, \ldots, 127\}$;
   - velocity, показывает как громко была сыграна нота $\{0, 1, \ldots, 127\}$.

2. Note off —- указывает на то, что нота закончила играть. В этом случае velocity показывает, как быстро нота была отжата.

Пример: “0, Note on, 0, 60, 50” означает “On channel 1, start playing a middle C with velocity 50 at time 0”.

Пример: “192, Note off, 0, 60, 20” означает “On channel 1, stop playing a middle C with velocity 20 at time 192”.

Каждое событие обладает меткой времени с момента начала либо в секундах, либо тиках.

2.1.4 Piano roll

Представление музыкальной композиции в виде Piano roll является, наверное, самым интуитивным. Как показано на (Рис. 16), сразу можно увидеть общую картину произведения.

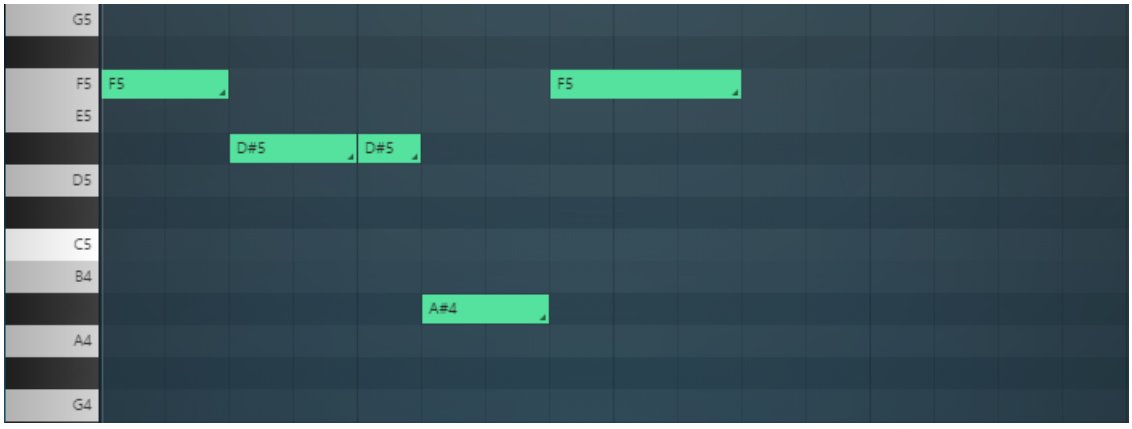

Рис. 16: Пример визуального представления piano roll, взятый из https://audionodes.com/docs/getting-started/composing-melodies/simple-melody/.

По оси x — время в секундах, по оси y — классы нот. Зеленая полоса — нажатая нота, длительность полосы, определяет длительность ноты.

2.1.5 ABC notation

ABC notation - представление музыки в виде текста (Рис. 17). Первые несколько строк содержат *мета-информацию*, такую как: название композиции (T), тональность (K), стандартную длительность ноты (L), размер (M) и т.д. Затем идет основой текст музыкальной композиции. Такое закодированное представление соблюдает некоторые основые принципы:

- Класс ноты, который закодирован латинской буквой (например: A соответствует Ля);
- Чтобы обозначить октаву используют специальные символы такие как «’» или пишут заглавную букву;
- Длительность обозначается «/» и последующей цифрой, если длительность ноты меньше стандартной или просто цифру без «/», если длительность выше стардартной;
- Такты разделены «|».

Задача генерации музыки в данном случае схожа с задачами NLP — natural language processing, изучающим проблемы анализа и синтеза естественных языков. Текст ABC notation обычно преобразуют в токены и составляют словарь токенов.

```
X:1
T:The Legacy Jig
M:6/8
L:1/8
R:jig
K:G
GFG BAB | gfg gab | GFG BAB | d2A AFD |
GFG BAB | gfg gab | age edB |1 dBA AFD :|2 dBA ABd |:
efe edB | dBA ABd | efe edB | gdB ABd |
efe edB | d2d def | gfe edB |1 dBA ABd :|2 dBA AFD |]
```

Рис. 17: Пример представления ABC notation, взятый из https://en.wikipedia.org/wiki/ABC_notation.

Такая нотация может представлять только монофонические мелодии, так как нет информации о других инструментах.

2.2 Генерация последовательностей

Универсальный способ генерации последовательностей данных с применением методов глубокого обучения заключается в обучении сети для прогнозирования следующего *токена* или следующих нескольких токенов в последовательности, опираясь на предыдущие токены.

Как обычно, при работе с текстовыми данными в роли токенов часто выступают слова или символы, и любая сеть, моделирующая вероятность появления следующего токена на основе предыдущих, называется *языковой моделью*. Языковая модель фиксирует *скрытое пространство языка* — его статистическую структуру.

После обучения такой модели: передав ей начальную последовательность, модель сгенерирует следующий токен.

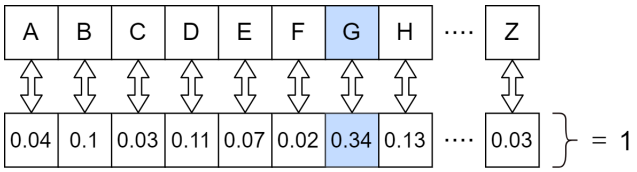

Рис. 18: Выбор наиболее вероятного значения следующего токена.

Затем добавит сгенерированный вывод в конец предыдущих входных данных и повторить процесс много раз (Рис. 18, 19).

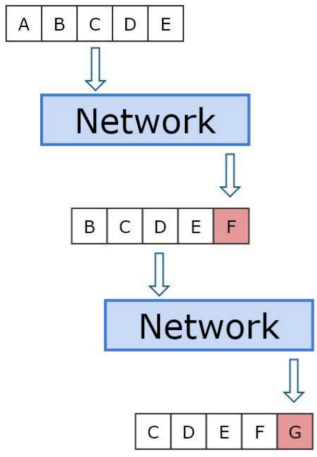

Рис. 19: Наглядное представление генерации следующего токена из [21].

Этот цикл позволяет генерировать последовательности произвольной длины, отражающие структуру данных, на которых обучалась модель.

Глава 3. Обзор существующих моделей

Выбор типа данных, способа их обработки и архитектуры влияет на то, какие инструменты нужно использовать для решение поставленной задачи. В данной главе будут рассмотрены некоторые из существующих архитектур для генерации музыки, каждая из них обладает своим уникальным подходом и результатами.

Для каждой из представленных моделей, предлагается рассмотреть три главные компоненты:

- Входные данные и их обработка;
- Архитектура сети;
- Результат.

3.1 Character-based RNN

Рекуррентные сети входят в состав архитектур большинства модей для генерации музыки. В модели, которая рассматривается в этой части, авторы предлагают подход на основе только RNN [21].

3.1.1 Входные данные и их обработка

В качетсве данных использовались midi файлы, соответствующие саундтрекам известной серии компьютерных игр «Final Fantasy». Все файлы были подвергнуты обработке, чтобы выделить из них информацию, используемую для обучения сети.

Каждый midi файл представляет собой последовательность событий включения и выключения нот. Авторы преобразовали их в последовательности элементов, каждый элемент, которых представлял собой ноту или аккорд (Рис. 20) с помощью библиотеки *Music21*. В такое представление они не включили информацию о длительностях нот и их громкостях, считая их фиксированными числами.

Далее, авторы выделили все уникальные ноты и аккорды, которые использовались во всех последовательностях, что дало им *словарь* всех

<music21.note.Note F>
<music21.chord.Chord B-2 F3>
<music21.note.Note E>
<music21.chord.Chord B-2 F3 >
<music21.note.Note D>

Рис. 20: Пример части полученной последовательности.

звуков и созвучий. С его помощью каждому элементу последовательности можно сопоставить целое число, соответствующие его номеру в словаре. Это используется для представления данных в **one-hot** виде.

One-hot представление: вместо последовательности нот и аккордов, теперь есть вектора размерности N, где N — размер словаря. Каждый вектор содержит N-1 нулей и единицу на той позиции, которая соответствует номеру ноты или аккорда в словаре. Все последовательности были представлены в виде таких векторов, для дальшейней обработки их нейронной сетью.

3.1.2 Архитектура

Архитектура данной модели состояла из:

- LSTM: был выбран по причине того, что для решения данной задаче крайне необходимо было дать сети возможность запоминать информацию.
- Dropout: слой, выполняющий регуляризационную функцию и уменьшающий переобучение сети за счет исключения определенного количества значений нейронов на случайных позициях.
- Dense: полносвязный слой, с последующей функцией активации для предсказания слежующей ноты или аккорда.

3.1.3 Результат

Авторы признают, что их сеть имеет множество органичений. Модель ничего не знает о существовании длительности нот и скорости их нажатия,

о паузах, о том, что может использоваться сразу несколько инструменов, а так же то, что существует куда больше нот и их созвучей, чем было в их словаре.

К сожалению, авторы не предоставили никакого анализа результатов, но можно послушать фрагметы, сгенерированные их моделью[1]. Стоит заметить, что сеть уловила некоторые музыкальные паттеры и усложнение архитектуры может привести к улучшению результатов.

3.2 MusicVAE

В [22] представлена модель под названием MusicVAE, нацеленная на создание коротких музыкальных композиций. Особенность этой модели в том, что она совмещает в себе VAE и RNN.

3.2.1 Входные данные и их обработка

Корпус данных состоял из MIDI файлов трех видов:

- Монофонические мелодии.
- Партии перкуссии.
- Композиции с тремя инструментами.

Модель тренировалась на разных данных, с разным количеством тактов. Для простоты будут рассмотрены только монофонические мелодии по 2 такта.

При обработке все файлы были квантизированы до 16 ноты. Это значит, что все ноты были смещены до ближайшей позиции музыкальной сетки с заданным шагом (16 разделений такта). То есть 2 такта были закодированы матрицей 32×130, где 32 - количество моментов времени, и 130 - количество состояний на каждый момент. Первые 128 позиций отвечают за то, какая нота нажата, 129 позиция отвечает за то, нажата эта нота или удержана, а 130 позиция отвечает за паузу.

[1] https://soundcloud.com/poush12/music-lstm-1

3.2.2 Архитектура

Для работы с последовательностями MusicVAE использует двунаправленную LSTM для кодировщика, и 2-х уровневую LSTM для декодировщика. Наличие непрерывного, скрытого пространства опеспечивает VAE.

Помимо сэмплирования из него, это скрытое пространство позволяет производить постепенную интерполяцию от одного музыкального фрагмента к другому.

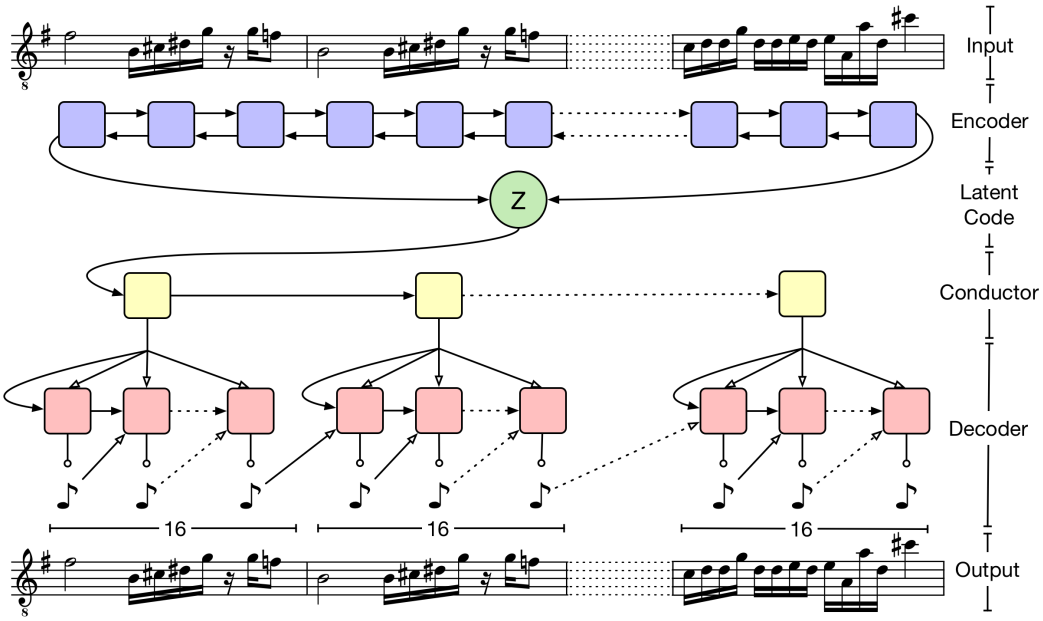

Рис. 21: Схема архитектуры MusicVAE.

Схема архитектуры (Рис. 21):

- *Encoder* — двунаправленная LSTM сеть. Последние скрытые слои двух направлений конкатенируются и передаются в последующий полносвязный слоей для инициализации скрытого пространства.

- *Decoder* — двух — уровневая иерархическая реккурентная сеть, которая и отличает decoder MusicVAE от обычных «flat» decoder'ов:

  - **Первый уровень (conductor)**. Латентное представление z проходит через полносвязный слой для инициальзации состояний первой ячейки LSTM, на вход которой подается вектор нулей.

После этого выход и скрытое состояние передается следующей ячейке.

– **Второй уровень (decoder)**. Состоит из нескольких LSTM, количество которых определяется длиной первого уровня. Состояния LSTM первого уровня инициализируют соответствующие LSTM второго уровня (т.е. состояния первого блока conductora'a передаются первому LSTM decoder'a и т.д.). Также стоит отметить, что каждый блок conductor'a отвечает только за свой LSTM decoder'a.

3.2.3 Результат

Пример интерполяции между двумя музыкальными фрагментами (по два такта каждый) можно увидеть на (Рис. 22).

Рис. 22: Интерполяция от левого до самого фрагмента за 14 шагов (28 тактов).

Для оценки модели авторы предоставляли людям по два фрагмента мелодий по 16 тактов (∼30с.). Они просили оценить музыкальность этих фрагментов по шкале Лайкерта, чтобы сравнить показатели между «flat» декодером, иерархическим декодером и настоящей музыкой.

В общий набор входили фрагменты всех трех видов архитектур (архитектрура для монофонических мелодий, архитектура для мелодии, баса и перкуссии и архитектура для перкуссии). Музыкальные фрагменты доступны для прослушивания[2].

Был собрано 192 мнения и по результатам опроса (Рис. 23) можно судить что во всех трех случаях люди больше предпочитают музыку, сгенерированную 2-х уровневым иерархическим декодером чем «flat» декодером, и партия перкуссии MusicVAE показывает незначительно лучшие результаты, чем настоящие фрагменты.

[2]https://storage.googleapis.com/magentadata/papers/musicvae/index.html

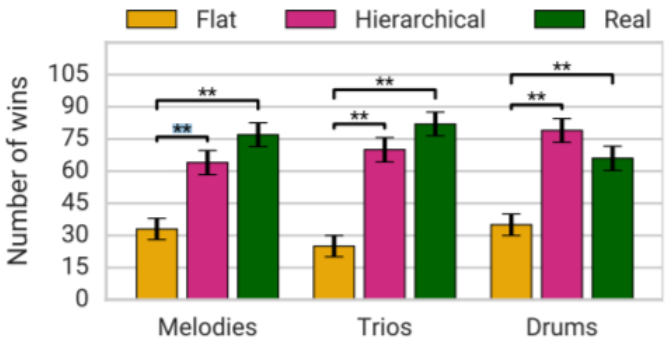

Рис. 23: Рузультаты опроса людей. Черные линии на барах показывают стандартное отклонение от среднего. Двойные звездочки над барами указывают на существуенно различие межны оценками.

3.3 WaveNet

WaveNet — нейросетевая архитектура для генерации аудио сигналов, которая была предложена в 2016 году [23]. Модель хорошо себя показала как в задачах синтеза человеческой речи, так и генерации музыки.

3.3.1 Входные данные и их обработка

Звуковая волна закодирована вектором $x = \{x_1, \ldots, x_T\}$, где x_i — в общем случае 16-битное целое число, соответствующее значению амплитуды в i-й момент времени. Частота дискретизации (количество замеров амплитуды в секунду) обычно равна 48кГц. Совместная вероятность всей волны описывается как произведение условных вероятностей

$$p(x) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}). \tag{1}$$

Модель учится предсказывать значение волны в момент времени t и сравнивает свои предсказания с реальным значением x_t, используя перекрёстную энтропию, как функцию потерь. Следовательно задача сводится к обычной многоклассовой классификации для каждого шага на протяжении всей волны.

Для задачи генерации музыки использовались два датасета:

- MagnaTagATune датасет состоял из 200 часов музыки. Каждый му-

зыкальный фрагмент длился 29 секунд и обладал метками жанра, инструмента, темпа, громкости и настроения.

- YouTube датасет фортепианной музыки, состоял из 60 часов аудио, взятых из видео на YouTube.

3.3.2 Архитектура

Главным инструментом WaveNet являются сверточные сети. Авторы используют модификацию обычных сверток, путем объеденения двух методов:

- *Causual convolutions* — сверточная сеть, использование которой гарантирует, что предсказание $p(x_{t+1} \mid x_1, \ldots, x_t)$, выдаваемое моделью в момент времени t, зависит только от предсказаний в предыдущие моменты времени (Рис. 24). Сама по себе архитектура представляет собой несколько подряд идущих сверточных слоев, где выход одного слоя становится входом для следующего.

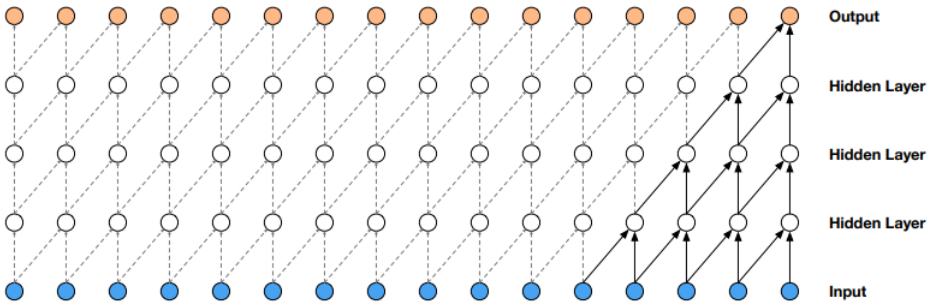

Рис. 24: Causual convolutions: зависит только от предыдущих моментов времени.

- *Dilated convolutions* — сверточная сеть, в которой фильтр применяется с некоторым заданным шагом, тем самым пропуская некоторые входные значения (Рис. 25). Это позволяет увеличивать размер рецептивного окна (в сверточных сетях — часть данных, которая «видна» фильтру в момент времени) с помощью небольшого количества слоев.

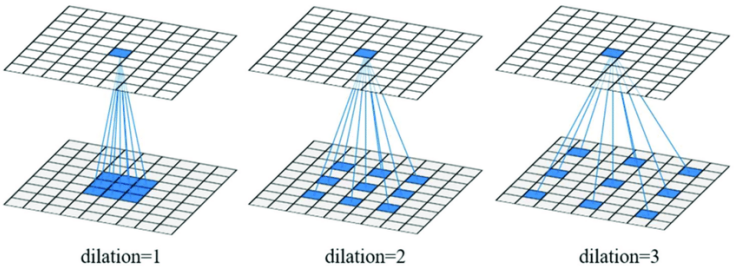

Рис. 25: Dilated convolutions: размер рецептивного окна при одинаковом ядре свертки (3×3) и разных значениях dilation [24].

Путем объединения двух таких подходов авторы получили сверточные слои, стеки которых позволяют достигать широкого рецептивного окна, сохраняя качество входных данный на протяжении всей сети. А так же получили возможность делать предсказания, основываясь только на предыдущих моментах времени (Рис. 26).

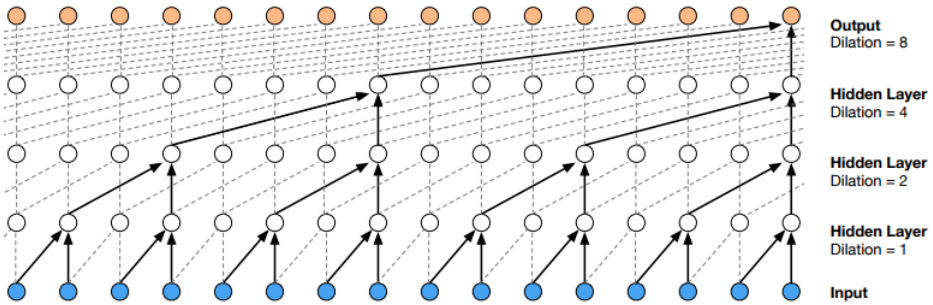

Рис. 26: Dilated causal convolutions: Хотя число нейронов, влияющих на каждый последующий нейрон, остается неизменным (2), каждое выходное значение напрямую зависит от всех входных. Кроме того, размер рецептивного окна увеличивается экспоненциально с увеличением количества слоев.

Как было сказано ранее, обычно аудио представлено в виде последовательности 16-битных целых чисел, следовательно выбор производится из 65536 возможный значений. Для уменьшение размерности распределения до 256 значений, авторы применили компандирование по μ-закону

$$f(x_t) = sign(x_t)\frac{\ln(1+\mu|x_t|)}{\ln(1+\mu)},$$

где $-1 < x_t < 1$ и $\mu = 255$. Как выяснилось, такое преобразование не сильно сказывается на качестве данных при их реконструкции.

В качестве активации, используется *gated activation unit*

$$z = tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x),$$

где $*$ обозначает операцию свертки, $\odot$ — поэлементное умножение, $tanh$ — гиперболический тангенс, σ — сигмоида, k — номер слоя, f и g обозначают фильтр и gate соответственно, а $W_{f,k}$ и $W_{g,k}$ — обучаемые сверточные параметры. Авторы утверждают, что для генерации аудиосигналов, такая функция активации значительно лучше, чем relu.

В архитектуре были использованы skip (соединение в нейросети, которое пропускает один или несколько слоев и передает информацию следующему слою) и residual (соединение в нейросети, которое передает информацию предыдущему слою) соединения для ускорения сходимости и возможности обучения более глубоких моделей (Рис. 27).

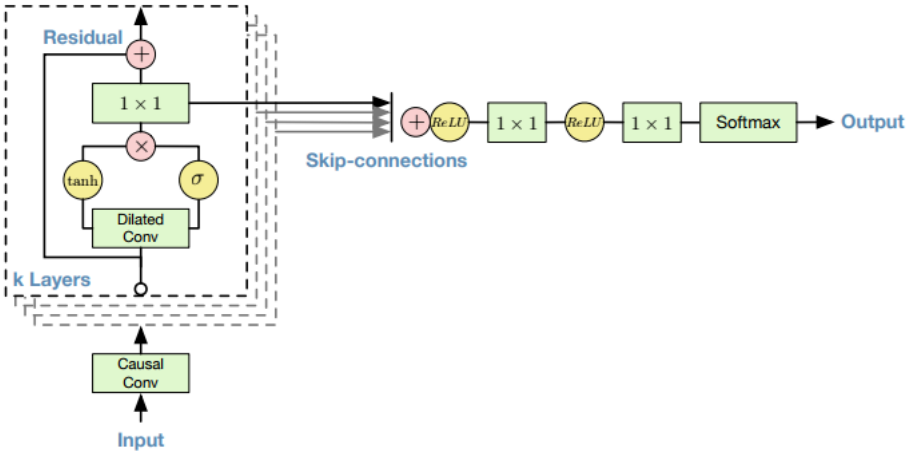

Рис. 27: Архитектура WaveNet.

Так же архитектура WaveNet позволяет добавлять условия. То есть, во входном веторе выделяется дополнительный слот h (условие) для возможности создания условного распределения $p(x|h)$ аудио по этому входу. Теперь (1) примет вид

$$p(x|h) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}, h).$$

Наложение таких условий, сподвигает WaveNet к генерации аудио

со специфическими характеристиками. Например при генерации музыки, можно передавать специальный тэг жанра или инструмента.

Условия бывают двух видов:

- Глобальные: входной параметр h фиксированный и влияет на финальное распределение во все моменты времени.
- Локальные: входной параметр h является специфическим для разных моментов времени.

3.3.3 Результат

Авторы провели множество исследований этой модели в рамках задачи синтеза речи. Что касается музыки: авторы также использовали эту модель для ее генерации. К сожалению, они не предоставили никакой объективной метрики, но провели субъективную оценку. Их главный вывод заключается в том, что широкое рецептивное окно имеет решающее значение для получения хороших результатов. Сгенерированные музыкальные отрывки были гармоничны и эстетически приятны, но им не хватало долгосрочной связности. Также были использованы глобальные условия жанра и инструмента. Результаты[3] получились многообещающие.

3.4 WaveGAN/SpecGAN

Представленные в 2018 году архитектуры генеративно-состязательных сетей WaveGAN и SpecGAN [26], способны синтезировать аудио данные. Структура этих сетей была позаимствована из более ранней работы [25], где в сети под названием DCGAN использовались сверточные слои в генераторе и дискриминаторе.

3.4.1 Входные данные и их обработка

Выход генератора в DCGAN (Рис. 28) — изображение 64 × 64 пикселя, что эквивалентно 4096 элементам последовательности аудио. Авторы

[3]https://storage.googleapis.com/magentadata/papers/musicvae/index.html

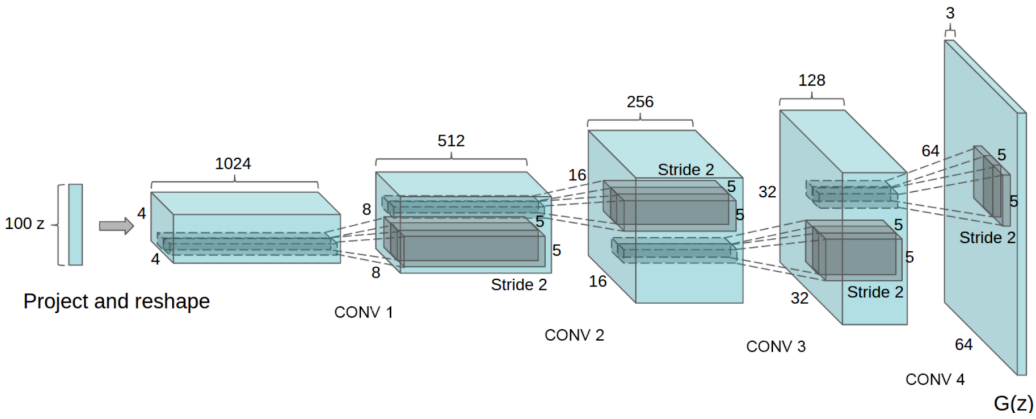

Рис. 28: Архитектура генератора DCGAN. На вход подается вектор z из латентного пространства. Он пропускается через несколько транспонированных сверточных слоев, каждый увеличивает представление в 4 раза (каждая сторона увеличивается в 2 раза).

добавили еще один транспонированный сверточный слой, для увеличения размерности до 1×16384 в случае WaveGAN или увеличения размерности до 128×128 в случае SpecGAN. При частоте дискретизации 16кГц, это эквивалентно примерно секундному аудио фрагменту.

Модели пробовали обучать на двух разных датасетах:

- Первый состоял из отрывков партий ударных по одному удару инструмента за раз. (∼45 минут)
- Второй состоял из отрывков фортепианной музыки. (∼20 минут)

3.4.2 Архитектура WaveGAN

В случае WaveGAN представление данных имеет вид вектора размерности размерности 1×16384. Это вызвано тем, что архитектура имеет дело не с 2D-изображениями, а с последовательностью значений амплитуды.

Оба фильтра на (Рис. 29) имеют одинаковое количество параметров и числовых операций. DCGAN использует небольшие (5x5), двумерные фильтры, в то время как WaveGAN использует более длинные (длиной 25), одномерные фильтры. При работе с 2D вектором, в каждом последующем транспонированном сверточном слове обе стороны увеличивались в 2 раза, что приводило к увеличению матрицы в 4 раза. Теперь при работе с 1D вектором, размерность повышается в 4 раза на каждом следующем слое.

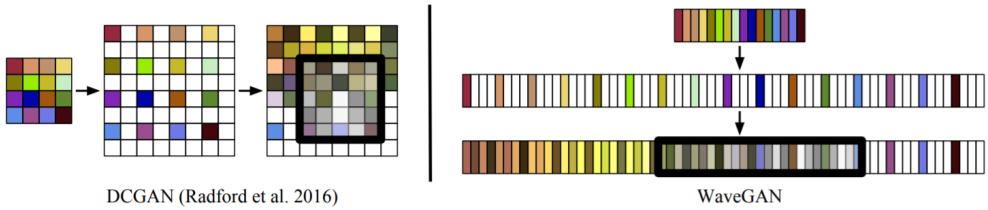

Рис. 29: Описание операции транспонированной свертки для первых слоев DCGAN (слева) и WaveGAN (справа).

Все это относится и к дискриминатору, который определен таким же образом. Используются фильтры длиной 25, а stride (шаг скольжения фильтра) увеличивается с 2 до 4.

Как известно, при работе с изображениями, транспонированные свертки могут добавлять артефакты (эффект шахматной доски) в выходные изображения. Проблема в том, что дискриминатор может уловить связь между этими эффектами на изображении и тем, что они были созданны генератором. Это может усложнить процесс обучения, так как дискриминатор будет с уверенностью отклонять сэмплы генератора.

Для аудио данных существует аналог таких артефактов. Высокочастотный шум, который может накладываться на частоты, обычные для реальных данных. Однако частоты артефактов всегда будут возникать на определенной фазе, что позволит дискриминатору изучить тривиальную задачу отклонения подобных генерируемых примеров.

Чтобы не дать дискриминатору научиться отыскивать такие эффекты, авторы предлагают операцию перемешивания фазы (phase shuffle) с определенным гиперпараметром n, которая случайным образом перемешивает фазы активаций каждого слоя на число из $[-n, n]$, дополняя недостающии места методом reflection padding (Рис. 30).

3.4.3 Архитектура SpecGAN

В случае SpecGAN, представление имеет вид спектрограммы, которое хорошо подходит для сетей GAN, создававшихся для генерации изображений. Кроме того, представление использует ту же размерность, что и WaveGAN (16384 образца дают спектрограмму размерностью 128×128).

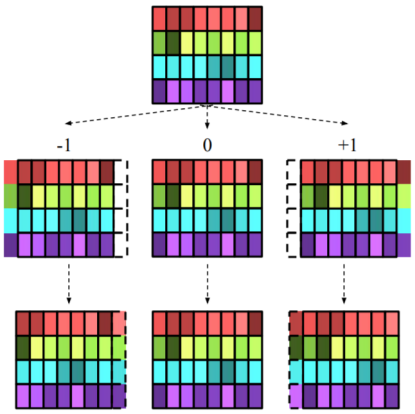

Рис. 30: Phase shuffle для n=1.

При обработки аудио данных, авторы сначала используют кратковременное преобразование Фурье с окном 16мс и шагом 8мс. Это дает им 129 возможных значений частот от 0 до 8кГц. Затем все значения частот были нормализированы с нулевым мат. ожиданием и еденичной дисперсией.

После таких преобразований, используется архитектура DCGAN, но как и в случае с WaveGAN был добавлен еще один слой для получения спектрограмм размера 128×128.

3.4.4 Результат

Были представлены результаты обоих моделей в задаче «Speech Commands Zero Through Nine» (SC09) синтеза слов от «zero» до «nine» на датасете, состоящим из записаных людьми соответствующих слов.

На (Рис. 31) показаны точности, с которыми люди угадывали слова на реальных записях, на записях сгенерированных WaveGAN и записях сгенерированных SpecGAN.

Metric	Real (test)	WaveGAN	SpecGAN
Human accuracy	.976	.943	.945

Рис. 31: Метрики аккуратности для Real data, WaveGAN и SpecGAN.

После прослушивания 10 сэмплов, каждый человек ставил субъективные оценки от 1 до 5 по критериям качества аудио сэмплов (quality), разборчивости (ease) и разнообразия (diversity).

Далее (Рис. 32) представлены средние оценки людьми quality, ease и diversity.

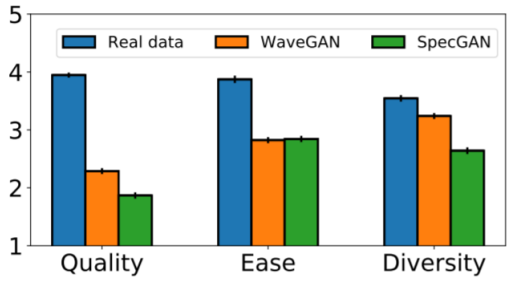

Рис. 32: Оценки quality, ease и diversity, предоставленные людьми.

На (Рис. 33) можно увидеть спектрограммы реальных данных, а также сгенерированных сэмплов WaveGAN и SpecGAN с использованием различных датасетов. Касательно музыки, нас интересуют drums и piano.

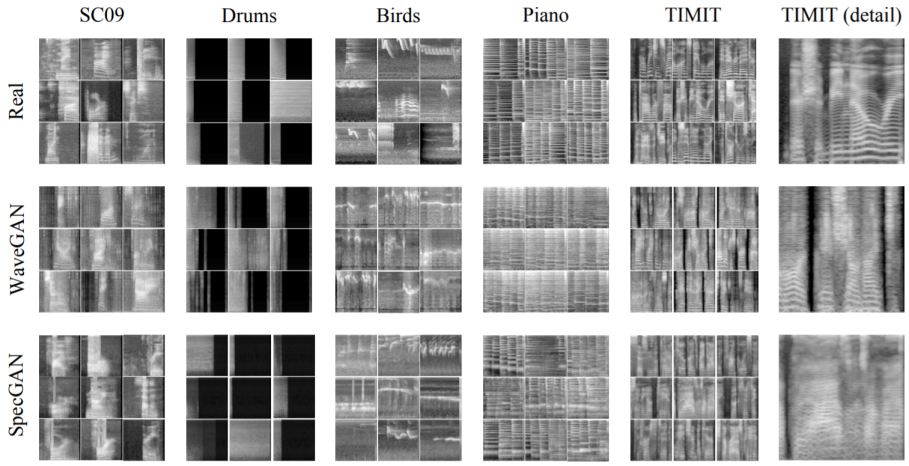

Рис. 33: Спектрограммы случайных сэмплов настоящих данных, WaveGAN, SpecGAN.

Видно, что WaveGAN справляется с синтезом речи куда увереннее SpecGAN, но все еще не достаточно хорошо, чтобы всерьез сравнивать его с реальными записями. Сгенерированные сетями аудио сэмплы находятся в общем доступе[4]. Подводя итог, можно сказать, что использование GAN для генерации аудио имеет большой потенциал. Особенно учитывая их успех в задачах генерации изображений.

[4] *https* : *//chrisdonahue.com/wavegan_examples*

Глава 4. Предлагаемая модель

Краткое описание данной главы:

- Будут предложены несколько вариантов схожих архитектур, для генерации музыки;
- Будут проведены эксперименты, сравнивающие разные модели;
- Также будут приведены и проанализированы результаты использования лучшей модели.

Модель, построенная в рамках работы, базируется на исследовании, описанном в 3.1.

4.1 Входные данные и их обработка

В отличие от способа обработки MIDI файлов в 3.1, было решено кодировать не только звуки и созвучия, но еще составлять два дополнительных словаря для их длительностей и velocity (громкостей). Это должно позволить модели понимать, как долго следует играть ту или иную ноту или аккорд, а также с какой громкостью их следует воспроизводить.

В качестве датасета были использованы MIDI файлы классической фортепианной музыки, соответствующие произведениям Моцарта и Бетховена. Всего было собрано 73 композиции, и после обработки получилось более 194 тысяч последовательностей нот и аккордов длины 100, и столько же для длительнотей и velocity.

Для каждого словаря строится векторное представление его элементов. После этого каждому элементу в словаре соответствует определенный вектор значений (word embedding) заданной размерности. Было решено использовать размерность вектора 100.

4.2 Архитектура сети

В рамках работы были построены три модели. Архитектуры всех трех моделей похожи и состоят из:

- *LSTM или Bidirectional LSTM слой* — рекуррентный слой нейронной сети, который принимает последовательность в качестве входных данных. Выход слоя состоит из последовательности векторов, соответствующих выходу каждой ячейки LSTM, либо из вектора, соответствующего выходу последней ячейки. В *Keras* это варьируется параметром *return_sequences*.

- *Attention слой* - реализует механизм внимания. На вход получает последовательности скрытых состояний с реккурентного слоя. Каждому вектору последовательности сопоставляет новый вектор, с примененным механизмом внимания.

- *BatchNormalization слои* — представляют собой метод для повышения производительности и стабильности нейронных сетей. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

- *Dropout слои* — это метод регуляризации, который исключает определённый процент (например 30%) случайных нейронов на слоях на разных итерациях (эпохах) во время обучения нейронной сети.

- *Dense слой* — полносвязный слой нейронной сети, где каждый вход соединен с каждым выходом.

- *Activation слой* — определяет какая функция активации будет использоваться для подсчета выходного значения. В работе была использована функция активации softmax.

4.2.1 Описание алгоритма

Расмотрим как работает данная модель:

1. На вход модели в каждый момент времени подается три последовательности по 100 элементов (нота/аккорд, длительность этой ноты

или аккорда, громкость). Каждый элемент каждой последовательности закодирован вектором действительных чисел рамерности 100 (word embedding).

2. Все тройки элементов, соответствующие одному моменту времени конкатенируются более длинные векторы длины 300 для дальнейшей обработки рекуррентным слоем.

3. На данном этапе на вход рекуррентному слою подается последовательность длины 100, каждый элемент которой состоит из объединенного вектора длины 300. Эта последовательность обрабатывается рекуррентным слоем и выдает последовательность скрытых состояних каждой ячейки.

4. К последовательности скрытых состояний применяеся механизм внимания, описанный в 1.4. На выходе имеется новая последовательность с примененным механизмом внимания.

5. Далее применяется еще один реккуррентный слой, выходом которого служит вектор, соответствующий значению последней ячейки.

6. Затем идут три полносвязных слоя, каждый отвечает за ноту/аккорд, длительность или громкость, с количеством нейронов равным количеству элементов в соотвествующем словаре. Каждый слой получает на вход один и тот же вектор с предыдущего рекуррентного слоя. Выходом каждого слоя является новый вектор значений, полученный путем домножения на соответствующую весовую матрицу и добавлением смещения.

7. К каждому из трех векторов применяется softmax функция активации, строятся категориальные распределения и выбираются наиболее вероятные элементы каждого распределения. Эти три элемента, соответствующие ноте/аккорду, длительности и громкости, подаются в функции потерь *categorical cross-entropy*, для сравнения их со своими метками. Ошибка прогноза передается по сети, для корректировки параметров.

Схематично это выглядит следующим образом (Рис. 34):

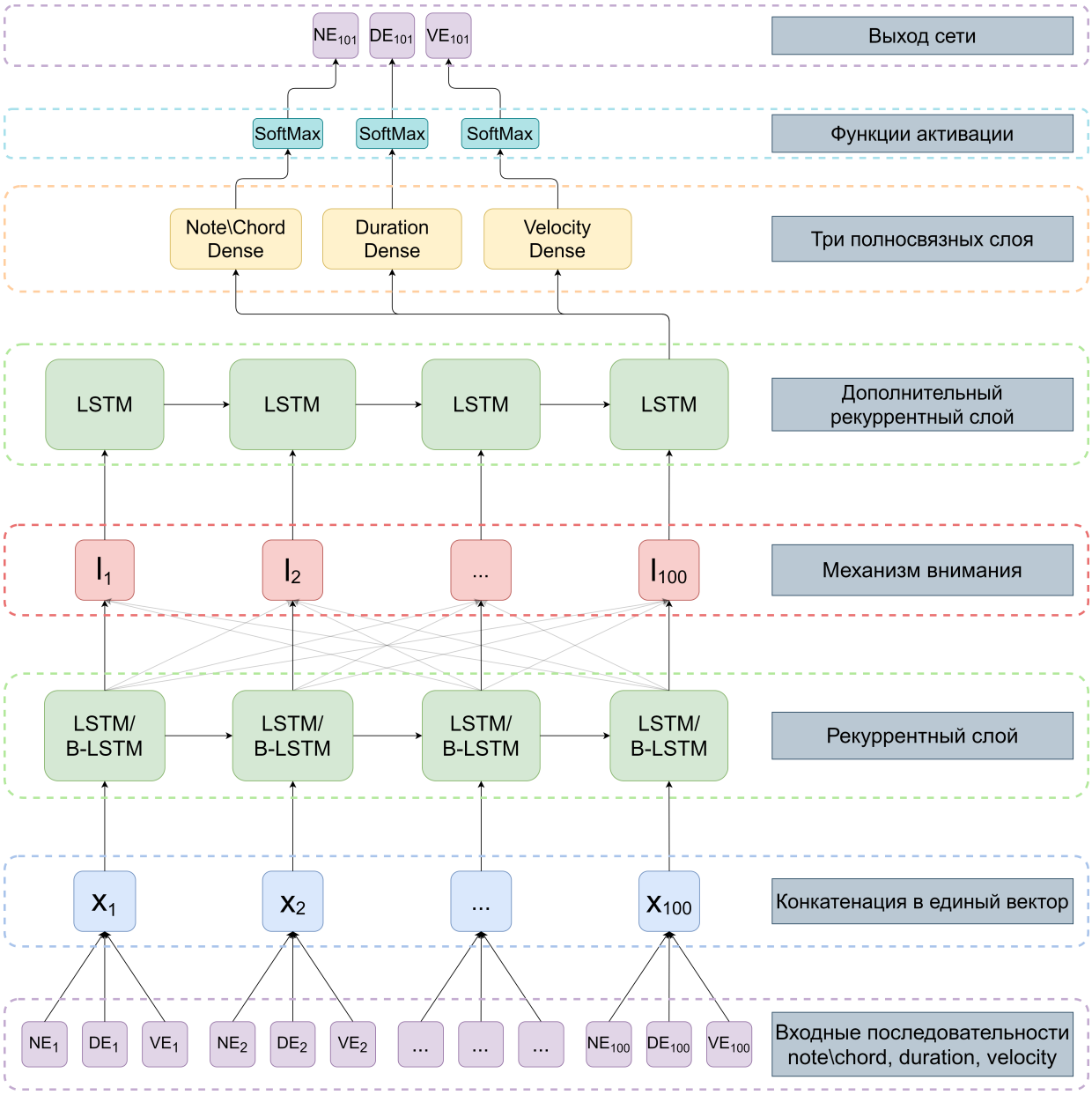

Рис. 34: Наглядное представление архитектуры сети. NE - последовательность embedding векторов нот/аккордов, DE - последовательность embedding векторов длительностей, VE - последовательность embedding векторов velocity.

Как было сказано ранее, в ходе исследования были построены три модели, отличавшиеся лишь первым рекуррентным слоем:

1. Первая модель в качестве первого реккурентного слоя имела LSTM слой с размером скрытого пространства 512.

2. Вторая модель в качестве первого реккурентного слоя имела связ-

ку из двух последовательных LSTM слев с размерами скрытых пространств 512.

3. Третья модель в качестве первого реккурентного слоя имела Bidirectional LSTM слой с размером скрытого пространства 512.

4.3 Параметры моделей

Для всех моделей использовались следующие параметры:

- Оптимизатор — Adam, Learning rate = 0.001.
- Функция потерь — категориальная кросс энтропия (categorical cross entropy), так как имеем дело с многоклассовой классификацией.
- Метрика — точность (accuracy), использовалась для визуального представления процесса обучения.

4.4 Результаты

Для каждой из характеристик (нота/аккорд, длительность, velocity) элемента последовательности применяется категориальная кросс энтропия, затем значения складываются, образуя общее значение потерь.

Ниже (Рис. 35) представлены графики сравнения сходимости общих потерь для каждой из моделей.

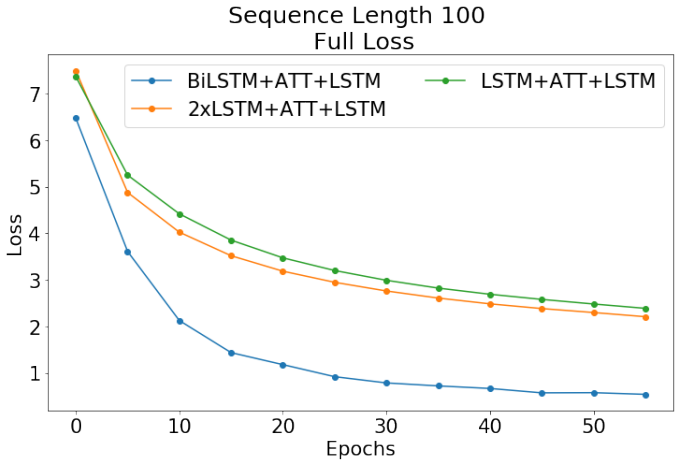

Рис. 35: Сравнение сходимости значений общих потерь.

Как видно из графика выше, можно сделать вывод, что лучше всего себя показал Bidirectional LSTM в качестве первого рекуррентного слоя.

Далее представлены значения функций потерь и значения точностей для каждой из характеристик в каждой модели (Рис. 36):

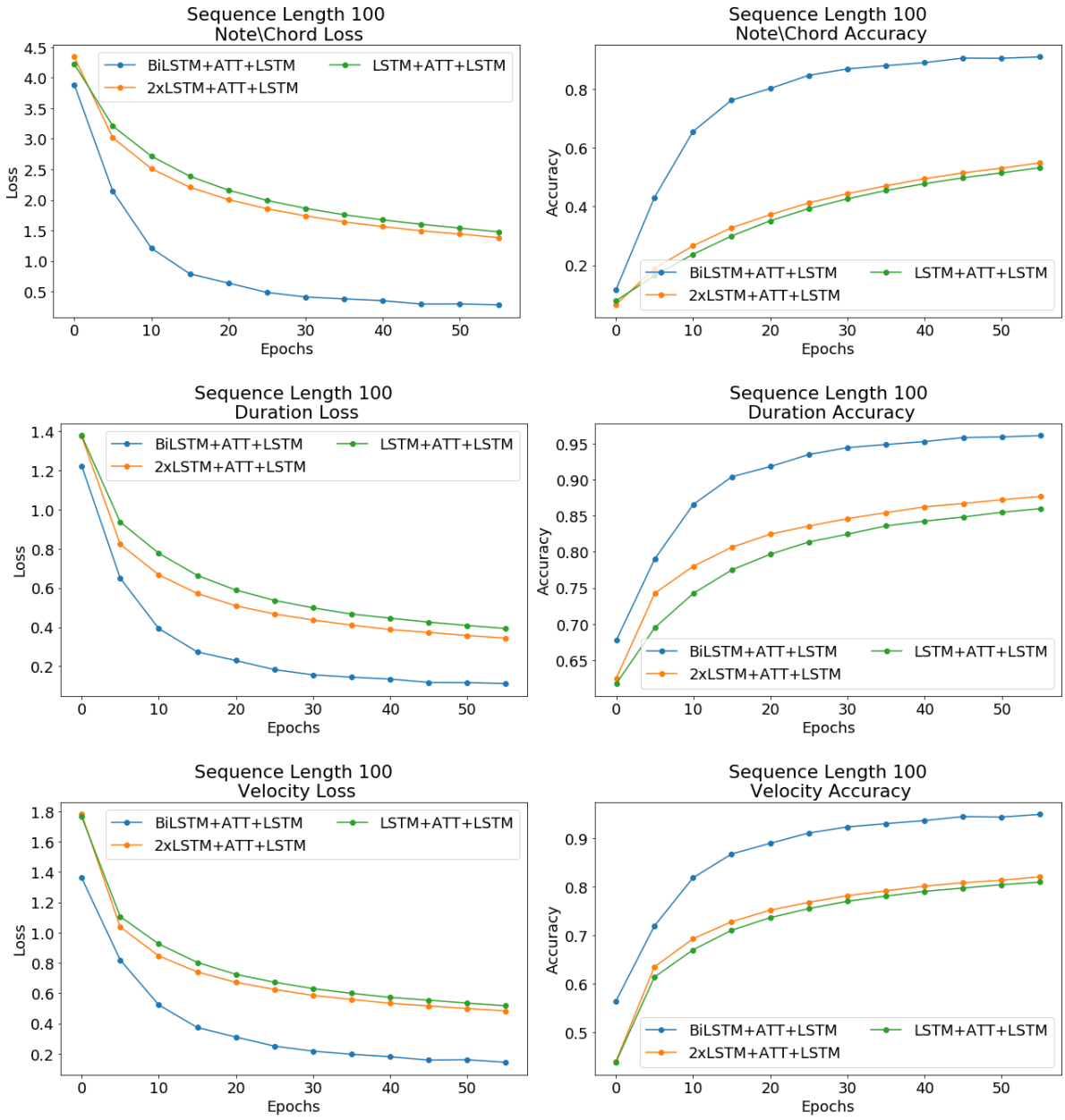

Рис. 36: Функции потерь и точности разных характиристик.

Все модели тренировались в течении 60 эпох, среднее время обучения составляло 12 часов на графическом процессоре.

Как и можно было ожидать, два последовательных слоя LSTM справляются с задачами лучше, чем один. Также можно заметить, что Bidirectional LSTM показывает лучшие результаты для всех трех характеристик.

4.4.1 Генерация и сэмплирование

После обучения всех моделей, для генерации была выбрана лучшая. Способ генерации описан в 2.2, за исключением того, что сэмплирование из вероятностного распределения происходило с помощью температуры, которая определяет «степень необычности» следующего токена. С учетом значения температуры и вероятностного распределения, вычисляется новое распределение, путем взвешивания вероятностей.

Без температуры процесс сэмплирования сводится к применения функции *softmax*. Пусть x_i — некоторый класс из $x = \{x_1, \ldots, x_n\}$, тогда

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}.$$

В то время как с температурой формула выглядит иначе

$$softmax_T(x_i) = \frac{e^{x_i/T}}{\sum_{j=1}^{n} e^{x_j/T}},$$

где T - температура.

Очевидно, что если $T = 1$, то распределение не продвергнется никакому изменению, однако, чем T больше, тем больше энтропия распределения вероятностей и тем более неожиданными и менее структурированными могут оказаться результаты. В том случае, если T близко к нулю, то величина случайной составляющей будет меньше и тем более предсказуемыми будут генерируемые данные.

Чтобы показать как влияет температура, была взята случайная последовательность из входного набора, и проведен процесс генерации следующих нескольких элементов (Рис. 37).

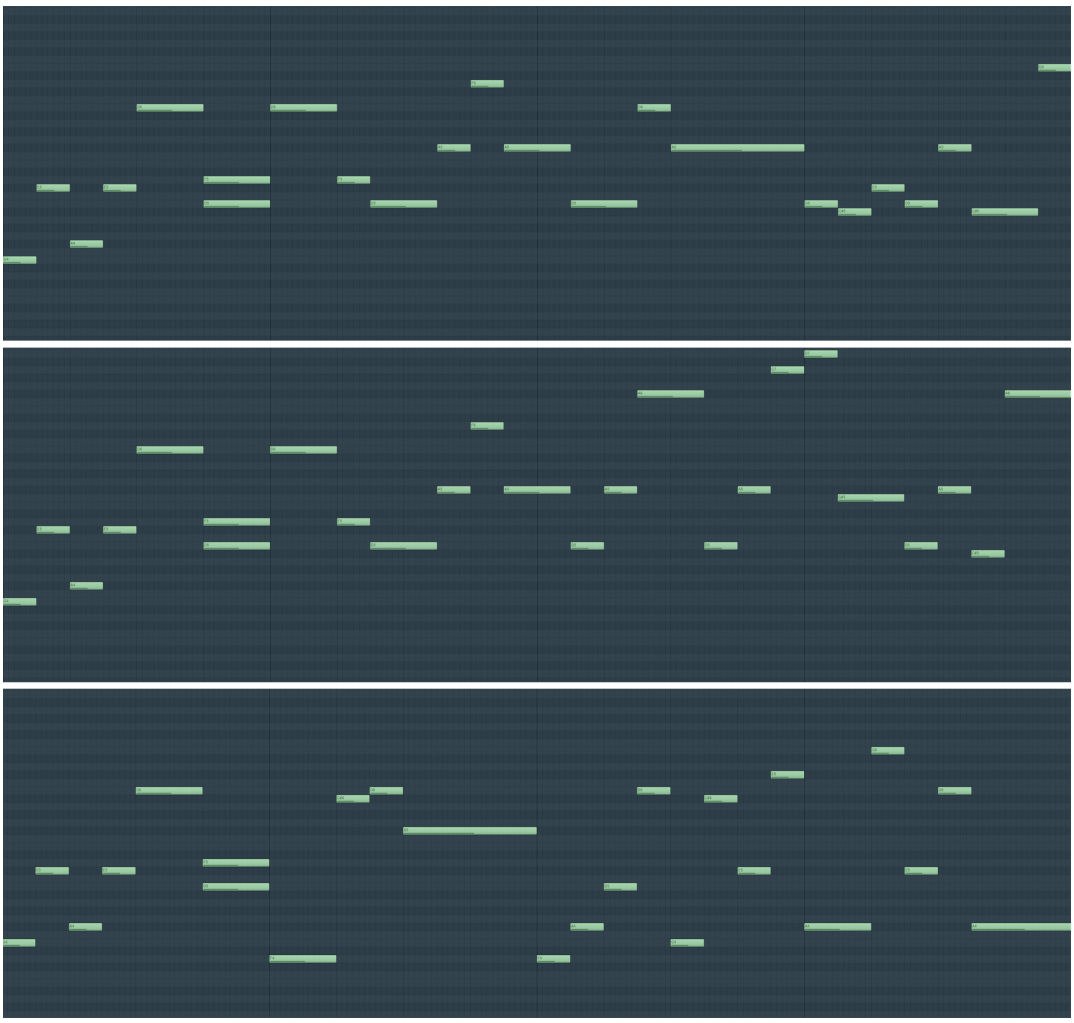

Рис. 37: Пример сгенерированных данных с разной температурой. Сверху T = 0.5, по центру T = 1.0, снизу T = 1.5.

4.5 Вывод

В результате использования всего 73 композиций модель научилась понимать некоторые музыкальные паттерны и улавливать связь между нотами и аккордами. Также стоит отметить, что сгенерированные музыкальные фрагменты обладают нотами разных длительностей и громкостей, что придает музыке динамичность и разнообразие. Примеры[4] сгенерированных фрагментов лежат в открытом доступе.

[4]https://soundcloud.com/user201801556/sets

4.6 Дальнейшие перспективы развития

Существуют несколько векторов дальнейшего развития:

- Увеличить объем обучающих данных, тем самым позволить модели узнать новые звуки и их комбинации в виде созвучий. Также это даст ей возможность уловить ранее не встречавшиеся паттерны построения музыки.
- Изменить входные данные с символьных на аудио. При этом придется использовать другие методы обработки. Это значит, что обучение будет происходить на совершенно по-другому устроенных последовательностях.
- Изменить архитектуру сети. Обратить внимание на другие, алгоритмы, такие как: GAN или VAE. Попробовать совместить их или использовать лучшие стороны каждой из моделей, если это возможно.

Заключение

В рамках исследования были проанализированы существующие методы генерации музыки. Также была описана и реализована архитектура сети. Путем изменения некоторых слоев, получены сравнительные показатели трех моделей. В результате была определена лучшая модель и с ее помощью были сгенерированы музыкальные фрагменты.

В выпускной квалификационной работе я ставил задачу разработать программный комплекс для генерации музыки и успешной ее решил.

Список литературы

[1] Briot J.-P. , Hadjeres G., Pachet F.-D. Deep Learning Techniques for Music Generation. Computational Synthesis and Creative Systems. / Springer, 1st ed, 2019. 284 p.

[2] François Chollet. Deep Learning with Python. / Manning Publications, 1st ed, 2017. 384 p.

[3] Sepp Hochreiter, Jurgen Schmidhuber. Long short-term memory // Neural Computation, Vol 9, No 8., 1997. P. 1735-1780.

[4] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks, 2015.

[5] Christopher Olah. Understanding LSTM Networks, 2015.

[6] J.-P. Briot, Francois Pachet. Music Generation by Deep Learning - Challenges and Directions. // Neural Computing and Applications, 2017.

[7] Ke Chen, Weilin Zhang, Shlomo Dubnov, Gus Xia, Wei Li. The Effect of Explicit Structure Encoding of Deep Neural Networks for Symbolic Music Generation. // International Workshop on Multilayer Music Representation and Processing (MMRP), 2019. P. 77-84.

[8] Luke Johnston. Using LSTM Recurrent Neural Networks for Music Generation, 2016.

[9] Allen Huang, Raymond Wu. Deep Learning for Music. // CoRR, Vol. abs/1606.04930, 2016.

[10] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. // IEEE Transactions on Signal Processing, Vol. 45, No. 11, 1997. P. 2673-2681.

[11] Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. // CoRR, Vol. abs/1409.0473, 2015.

[12] Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, Feifei Li. OpenTag: Open Attribute Value Extraction from Product Profiles. // Association for Computing Machinery, 2018. P. 1049–1058.

[13] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and timeseries. // The handbook of brain theory and neural networks, 1998. P. 255-258.

[14] G. E. Hinton, R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. // American Association for the Advancement of Science, Vol. 313, Issue 5786, 2006. P. 504-507.

[15] Diederik P. Kingma, Max Welling, Auto-Encoding Variational Bayes, 2013.

[16] Danilo Jimenez Rezende, Shakir Mohamed, Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. // PMLR, Vol. 32, 2014. P. 1278-1286.

[17] Carl Doersch. Tutorial on Variational Autoencoders, 2016.

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. // NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems, Vol. 2, 2014. P. 2672–2680.

[19] Bharath Ramsundar, Reza Zadeh. TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning. / O'Reilly Media, 1 ed., 2018. 256 p.

[20] Simon Dixon, Zhiyao Duan Member, Sebastian Ewert. Automatic Music Transcription: An Overview. // IEEE Signal Processing Magazine, Vol. 36, 2019. P. 20-30.

[21] Piyush Agrawal, Sajal Kaushik, Subham Banga. Automated Music Composition using LSTM, 2018.

[22] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, Douglas Eck. A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. // PMLR, Vol. 80, 2018. P. 4364-4373.

[23] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu. WaveNet: A generative model for raw audio, 2016.

[24] Ximin Cui, Ke Zheng, Lianru Gao 2, Bing Zhang, Dong Yang, Jinchang Ren. Multiscale Spatial-Spectral Convolutional Network with Image-Based Framework for Hyperspectral Imagery Classification. // Remote Sensing, Vol. 11, 2019. P. 2220.

[25] Alec Radford, Luke Metz, Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks // 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium, 2018. P. 31-38.

[26] Chris Donahue, Julian McAuley, Miller Puckette. Adversarial Audio Synthesis. // ICLR, 2018.